# Bridging the Gap between Software Process and Software Development

## Challenges in Model-Driven Engineering

**Emmanuelle Rouillé* **, Benoit Combemale**, Olivier Barais**, David Touzet* and Jean-Marc Jézéquel****

**Sodifrance, P.A. la Bretèche, avenue Saint-Vincent, 35768, Saint-Grégoire, France*
*{erouille,dtouzet}@sodifrance.fr*

***Université de Rennes 1, IRISA, Campus de Beaulieu, 35042, Rennes, France*
*{emmanuelle.rouille,benoit.combemale,olivier.barais,jean-marc.jezequel}@irisa.fr*

RÉSUMÉ. *L'Ingénierie Dirigée par les Modèles (IDM) bénéficie au développement logiciel (développement logiciel dirigé par les modèles) ainsi qu'aux processus logiciels (modélisation des processus). Pourtant, l'écart entre les processus et le développement est toujours trop grand. En effet, l'information des processus n'est pas toujours utilisée pour améliorer les développements et inversement. Par exemple, on peut définir dans une description de processus les outils de développment utilisés sans les relier avec les outils réels. Cet article motive le besoin de réduire l'écart entre les processus et le développement, grâce à l'IDM. Un processus logiciel industriel réel est utilisé à titre d'illustration.*

ABSTRACT. *Model Driven Engineering (MDE) benefits software development (a.k.a. Model Driven Software Development) as well as software processes (a.k.a. Software Process Modeling). Nevertheless, the gap between processes and development is still too great. Indeed, information from processes is not always used to improve development and vice versa. For instance, it is possible to define the development tools used in a process description without linking them to the real tools. This position paper illustrates the need for bridging the gap between software process and software development, using MDE. A real industrial software process is shown as an example.*

MOTS-CLÉS : *IDM, processus de développement logiciel.*

KEYWORDS: *MDE, Software Development Process.*

## 1. Introduction

Model Driven Engineering (MDE) benefits software development. It provides mechanisms to define modeling languages to express software systems at a higher level of abstraction in order to simplify their description. It also provides model to model transformations and code generation facilities. MDE also benefits software processes by providing languages dedicated to the management of processes (Bendraou *et al.*, 2009). However, we will illustrate in the following of this article the need for bridging the gap between software processes and development using MDE. The idea is to use process information to drive configuration, deployment and adaptation of development tools, to trace versions of software artifacts, to verify and to capitalize them. Inversely, development information can be used to adapt these processes. For instance, the modification of a development artifact can drive the adaptation of tools used in the process. In this context, we benefit from a collaboration with Sodifrance, a software and computing services company. It provides an industrial process example which illustrates current issues for software development.

This paper is organized as follows. In section 2 we present industrial trends in software processes for model-driven development. In section 3 we discuss current issues and challenges. In section 4 we highlight related work. Finally, in section 5 we conclude and present our perspectives.

## 2. Industrial Trends in Software Process for Model-Driven Development

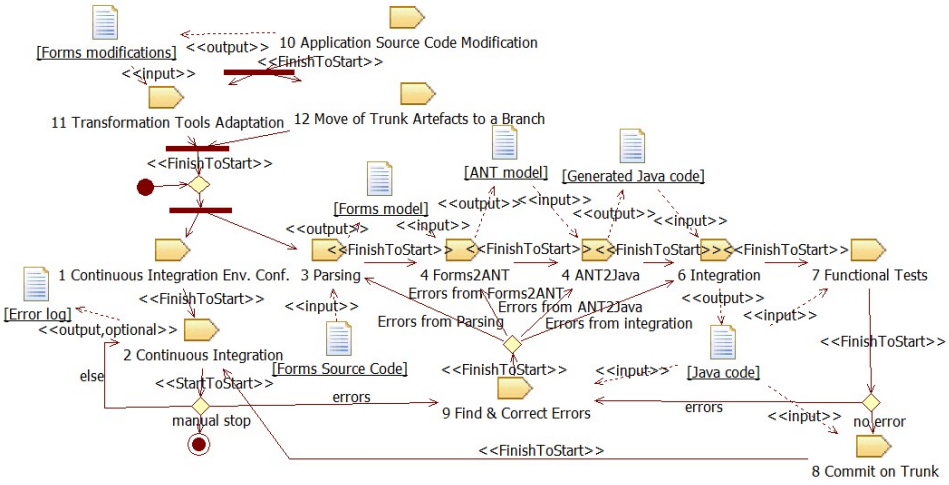


**Figure 1.** *Migration process*

In this section we describe a simplified model-driven development process from the Sodifrance company. It highlights current difficulties in this industry to implement

a software intensive system. It consists of migrating an Oracle Forms application to Java/J2EE. The functionalities of the migrated application have to be the same as these of the source application. There is no evolution. The process is described in Figure 1 and detailed in the rest of this section.

When initiating the migration of a consistent set of source code, a continuous integration support for the considered migration scope is configured (task 1) and launched (task 2). In parallel, the migration process is started. It consists of several automated transformations : Forms source code to Forms model (task 3), Forms model to ANT[1] (task 4), and ANT model to Java/J2EE code (task 5). Then an manual integration step is performed (task 6) : the Java code is corrected so that it compiles and functionalities that are not generated (because they have specificities that do not match with a general generation pattern and generating them would imply to add to much information to the ANT model) are implemented. Then, functional tests are executed (task 7). Errors on Java code can be detected during this step, when a functionality does not have the expected behavior. In this case, their origin is found, corrections are made, and the process is run again from the origin of the errors (task 9). If there is no error, the Java code is added to a version control system (task 8). During the migration process, continuous integration is performed as a background task, with the code under version control as input (task 2). If errors are detected, task 9 is performed. The source code of the application can also evolve once the migration process has started (task 10). Changes may include bug fixings as well as functional evolutions. These modifications need to be migrated so that the delivered application will be up to date. Thus, transformation tools may be adapted to integrate the modifications (task 11), the version of the previous application is copied on a version control system branch (task 12) and the process is run again. Note that the development process of transformation tools is not described here.

During these steps, multiple tools are used. There are development tools, such as transformation tools (a parser, MIA-Transformation, MIA-Generation[2]) and Eclipse IDE. There are also software project management tools (SVN version control system, Maven, Hudson, Selenium).

The process description highlights the following issues :

– *Heterogeneity of the tools* : a lot of tools are manually configured, deployed and adapted. This brings complexity.

– *Multiple versions of tools* : when a transformation tool is modified, a new version appears. This implies : multiple versions into the same range of tools, repercussions on tools depending on the modified one, difficulties in knowing which version corresponds to a specific version of a deliverable, in knowing differences between two versions and in knowing what are the existing versions and what they do. The last point leads to difficulties for reusing artifacts from one project to another.

---

1. ANT is a platform independent metamodel from Sodifrance, dedicated to web technologies. It represents static data structures, actions, algorithms, UIs, widgets and navigation.
2. MIA-Transformation and MIA-Generation are MIA-Software (`http://www.mia-software.com`) products to implement respectively model transformations and code generations.

– *Late verification of process artifacts* : only the final outcome (the Java code) of the process is verified, creating difficulties in finding from which step errors come because artifacts are transformed several times during the process.

– *Manual recurrent tasks*, such as continuous integration environment configuration, or the move of an old application version from the SVN trunk to a branch. These tasks are time consuming and error prone.

– *Modification of process inputs*, such as the client modification of the Forms code. This entails using process adaptation to manage the new version of these artifacts. This also entails running the process again.

## 3. On Current Challenges

In this section, we identify current challenges and show how they can address the issues raised in section 2.

– *Process-driven and requirements-based configuration, deployment and adaptation of development tools* : capturing tools and their characteristics into processes description addresses the complexity induced by their heterogeneity and manual recurrent tasks. This would allow their configuration, deployment and adaptation to be driven.

– *Process-driven and requirements-based configuration management* : during the process execution, linking versions of different tools between them, capturing differences between two versions of a tool, as well as capturing what the versions do, would address multiple versions of tools concern. Moreover, finding similar requirements in process descriptions would be a way to choose similar versions of tools from one project to another and so to improve reuse of artifacts.

– *Process-driven and requirements-based software artifacts verification* : early artifacts verification is a way to simplify errors diagnosis and to correct them faster. Capturing requirements in the process description would drive this verification.

– *Development-driven process adaptation* : a new version of a tool as well as modification of process inputs may lead to the adaptation of other tools. Process steps depending on adapted tools have to be run again, producing adapted process artifacts. This leads to process adaptation. Representing a process with a product line, that is a process line (Rombach, 2005), and automatically derive a process configuration, would be a way to automate process adaptation.

## 4. Related Work

In this section we show what challenges have been addressed in the literature and what challenges remain.

Software Process Modeling Languages (SPMLs) exist in the literature that allow a tool definition to be captured (Bendraou *et al.,* 2009, Scott *et al.,* 2001). However, there is no mechanism that allows configuration, deployment and adaptation of tools to be driven at process execution.

According to the configuration management, an extension of the AM3 framework [3] has been proposed in the Mopcom-I project [4] in order to capture a process configuration model. We now need a model versioning system in order to be able to retrieve a specific version of a process configuration model to produce a specific version of an artefact. Several approaches have been proposed (Altmanninger *et al.,* 2009). But none of them is at the same time independent of the metamodel, independent of the modeling tool and allows users to manage the granularity of the elements to version without modifying the metamodel to add version metadata. On the other hand, in (Koudri *et al.,* 2010), the authors propose an extension of the SPEM metamodel with the notion of intention. In (Konrad *et al.,* 2007), authors propose an approach to check that UML design models verify requirements captured in a goal model. But we still miss an SPML allowing to capture requirements.

In terms of process adaptation, approaches have been proposed to define and model process lines (Durán *et al.,* 2003, Hallerbach *et al.,* 2008, Rosemann *et al.,* 2007), to automate product derivation (Ziadi *et al.,* 2006) and to link product derivation to requirements (Than Tun *et al.,* 2009). However, none of them deal with the automated derivation of processes from requirements.

In (Liaskos, 2008), the author proposes a direction for connecting goals with configurations of software systems. Finally, contributions from the Mopcom-I project allow process configuration models to be automatically adapted according to a set of modifications.

## 5. Conclusion and Perspectives

We showed how MDE raises new possibilities for linking process description and execution to the software building. To our knowledge, challenges still need to be addressed in this area. The first is the consideration of tools configuration, deployment and adaptation at process execution. The second is the modeling of requirements into process description. The third is automated derivation of process configuration from requirements. As future work we plan to use adaptation abilities of development environments to drive their configuration, deployment and adaptation. In particular, we will reuse models@runtime facilities combined with dynamic process lines and requirements to improve agility in software development environments.

## 6. References

Altmanninger K., Seidl M., Wimmer M., A Survey on Model Versioning Approaches, Technical report, Johannes Kepler University Linz, 2009.

Bendraou R., Jézéquel J.-M., Gervais M.-P., Blanc X., « A Comparison of Six UML-Based Languages for Software Process Modeling », *IEEE TSE*, 2009.

3. http://www.eclipse.org/gmt/am3
4. http://mopcom-i.gforge.inria.fr/index.php?n=Main.HomePage

Durán A., Benavides D., Bermejo J., « Applying System Families Concepts to Requirements Engineering Process Definition », *PFE*, LNCS, 2003.

Hallerbach A., Bauer T., Reichert M., « Managing Process Variants in the Process Life Cycle », *ICEIS*, 2008.

Konrad S., Goldsby H., Cheng B. H. C., « iMAP : An Incremental and Iterative Modeling and Analysis Process. », *MoDELS*, 2007.

Koudri A., Champeau J., « MODAL : A SPEM Extension to Improve Co-design Process Models », *ICSP*, LNCS, 2010.

Liaskos S., Acquiring and Reasoning About Variability in Goal Models, PhD thesis, University of Toronto, 2008.

Rombach H. D., « Integrated Software Process and Product Lines », *ISPW*, 2005.

Rosemann M., van der Aalst W. M. P., « A configurable reference modelling language », *Information Systems*, 2007.

Scott L., Carvalho L., Jeffery R., D'Ambra J., « An Evaluation of the Spearmint Approach to Software Process Modelling », *Software Process Technology*, LNCS, Springer, 2001.

Than Tun T., Boucher Q., Classen A., Hubaux A., Heymans P., « Relating requirements and feature configurations : a systematic approach », *SPLC*, 2009.

Ziadi T., Jézéquel J.-M., *Software Product Lines*, Springer-Verlag, chapter Product Line Engineering with the UML : Deriving Products, 2006.